

lecture 20

binary search trees (1)

```

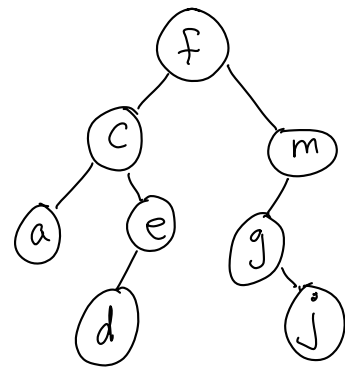
class BSTNode <K> {
    K key
    BSTNode <K> left
    BSTNode <K> right
}
    
```

Keys are comparable i.e. there is a strict ordering relation $<$

Binary Search Tree (Definition)

- binary tree
- elements (keys) are comparable
- each node has a unique key (two nodes have different keys)
- for any node, all descendants in left subtree are less than that node, and all descendants in right subtree are greater than that node

Example



Inorder traversal gives: a c d e f g j m

Enumerate all the BSTs with keys $\{a, b, c, d\}$.

First consider BSTs with "a" at root.

Then consider BSTs with 'b' at the root

Then consider BSTs with 'c' at root (similar to case 'b' above).

Then consider BSTs with 'd' at root (similar to case 'a' above).

Let $t(n)$ be the number of BSTs with n nodes.

$$t(n) = \sum_{i=1}^n t(i-1) \cdot t(n-i)$$

eg.

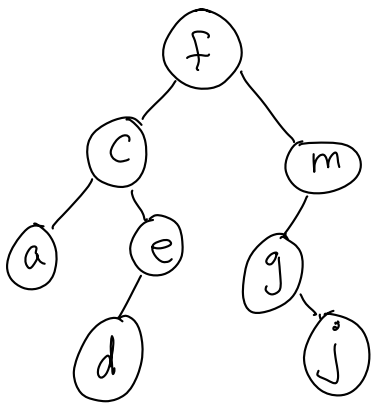
$$t(4) = t(0)t(3) + t(1)t(2) + t(2)t(1) + t(3)t(0)$$

where $t(0) = 1$
the empty tree

$t(1) = 1$
tree with one node

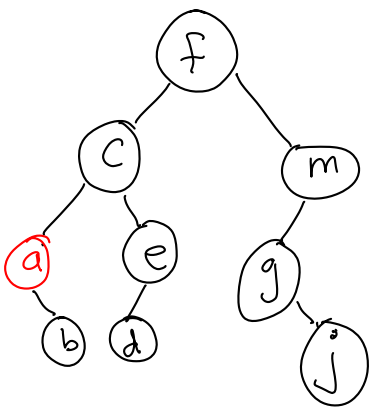
BST ADT

- find (key)
- findMin()
- findMax()
- add (key)
- remove (key)



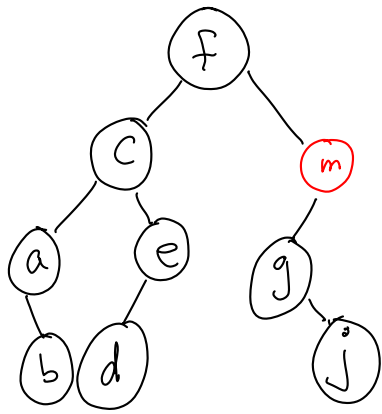
find (root, 'g')
find (root, 's') } return the node

```
find (root, key) { // return the node
  if (root == null or root.key == key)
    return root
  else { if key < root.key
    return find (root.left, key)
  else
    return find (root.right, key)
  }
}
```



findMin (root)

```
findMin (root) { // return the node
  if (root == null or root.left == null)
    return root
  else
    return findMin (root.left)
}
```



findMax (root)

```
findMax (root) {  
  if (root == null or root.right == null)  
    return root  
  else  
    return findMax (root.right)  
}
```